DL-based Support to Domain Engineering

Ernesto Compatangelo
School of Computer and Mathematical Sciences, The Robert Gordon University, Scotland — compatan@scms.rgu.ac.uk
Francesco M. Donini
Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Italy — donini@dis.uniroma1.it
Giovanni Rumolo
Dipartimento di Informatica ed Automazione, Università di Roma TRE, Italy — rumolo@inf.uniroma3.it

1 Overview and Motivations

We believe that an automated support tool for Domain Engineering (DE) should behave as an Intelligent Knowledge Management Environment (IKME) and not as an expert system. It should only help the analyst in the discovery and in the explicitation of contradictions, redundancies and hidden properties detected in the Domain Knowledge Base (DKB) under development and/or analysis. In other words, it should not suggest modifications to (or worse, directly modify) the description of concepts in the considered DKB.

Present IKMEs for Domain Engineering suffer, in our opinion, from their mixing up two representation levels:

1. a user-oriented conceptual level, where domain-specific elements are represented (e.g., data, processes and flows in Data-Flow Diagrams [9, 11]);
2. an underlying logic-oriented epistemological level, where inferences are drawn from general-purpose elements (e.g., DL concepts and roles).

The explicit separation of operational tasks between humans and computer systems should imply a corresponding separation of levels (1) and (2) in support tools.

We claim that in most of present IKMEs one of the two above levels is not explicitly present, i.e., it is "embedded" in the other level. A few IKMEs (e.g., [1]), directly implement specialized reasoning procedures based on the distinct conceptual primitives of the modelled domain, such as entities, relationships and so on. Conversely, the majority of IKMEs (e.g., [6, 2] directly "program" general-purpose reasoners[1]. In other words, they describe domain knowledge in terms of epistemological primitives (such as concepts and roles, sets and relations, nodes and arcs and so on). We believe that both approaches suffer from some drawbacks. In the direct implementation of specialized reasoning procedures, only a particular concept model is supported. This means that variants of the (already existing) reasoning algorithms of general-purpose reasoners have to be re-implemented. In the direct usage of general-purpose systems, the conceptual level is implicit, making it difficult to assess the representation adequacy. Moreover, since modelling is not performed in terms of domain-specific concepts, an intelligent explanation of drawn inferences, which is part of the support, must be separately built.

We propose an engineering approach to the development of IKMEs for conceptual modelling and analysis which explicitly deals with both representation levels at the same time. This gives rise to a *multilevel approach* which is more suitable from an engineering perspective. More precisely, a support tool built following our approach allows for the explicit domain-dependent description of instances of different conceptual primitives. Each primitive is endowed with a specification of its allowable set of reasoning services. User-oriented descriptions are transformed into a corresponding set of logic-oriented ones, as well as domain-specific inferences are mapped into general-purpose reasoning services. In this way, we obtain IKMEs that can capture several widespread exemplary concept models at once, performing at the same time domain-specific KR&R at the user level. Different application domain aspects described in terms of the DE models listed below can be thus merged and analysed in a more uniform and integrated way. Popular DE models include both basic and extended entity-relationship models (E-R and EER, [8]), functional models in Data Flow-based Structured Analysis (SA, [11]) as well as in the functional part of Object Modelling Technique (OMT [9]) and object models (e.g., object-centered approaches or the object part of OMT).

Extending the approach introduced in [4], we exemplify our proposal by modelling Data-Flow Diagrams in level (1) and by using a DL with a Closed Terminology Assumption (CTA) [10] in level (2). The benefit of our approach lies in decoupling the conceptual level from the underlying epistemological level, thus allowing for a more open architecture of support tools for the construction and the analysis of large domain models.

[1] Here and in the following, general-purpose reasoners are also called inferential engines or KR systems.

2 Engineering of IKMEs – our proposal

The engineering approach proposed in this paper for the development of IKMEs explicitly deals with both representation levels at the same time. It allows the domain-dependent description of concepts in terms of different syntactical constructors (i.e. meta-concepts), each endowed with its set of specialized inferences. These user-oriented concept descriptions are transformed into a corresponding set of logic-oriented ones, as well as domain-specific inferences are mapped into general-purpose ones.

A first consequence of the introduction of a multi-level approach dealing with the two cited levels is the need for an explicit structured description of concepts. These structured descriptions should not be freely arranged, but a different syntactical constructor should be introduced for each relevant group of domain concepts (e.g. one constructor for data, one for processes, etc.). Each syntactical constructor corresponds to a fundamental abstraction elicited in the application domain.

A second consequence of the introduction of a multi-level approach is the need for a correspondence with the underlying domain-independent level. This correspondence can be either achieved introducing new inferences in a programmable environment, such as in PROTODL, or introducing *emulators*, as in our proposal.

Domain Engineering activities involve some particular findings which were explicitly considered during the development of our approach. Under a user-oriented perspective, restrictions of KR languages only driven by worst-case analysis are not useful. Although the introduction of some language constructors can change the inference problem, thus requiring a worst-case exponential time, a disciplined extension of the language gives rise to a less demanding inference problem. During domain modelling, analysts introduce a large number of concepts which are non-expressible because of language restrictions. These concepts become primitive ones for the IKME, i.e., they become a *fake* [7] for domain analysis as well as for the classification algorithm. Indeed, we believe that some reasonable *fakes* are necessary as not every concept needs to be fully represented. We thus introduce highly structured syntactical constructors to represent domain concepts in a straightforward way. These constructors play a strategic role in the achievement of an effective domain level.

The vagueness involved in conceptual activities, mainly due to incomplete knowledge [9], gives rise to a serious question about the capability of DE in supporting systems which tolerate incomplete entailment. We firmly maintain that completeness and correctness must be granted for every reliable inference system.

As DE does not require fast answers, but *inference utility* [7], our approach explicitly deals with domain-specific deductive services. Therefore, we are much more interested in defining very useful inferences for a specific application domain, than in obtaining slightly useful fast responses. For instance, we consider inferences drawn from a data flow assertion as a highly useful service for domain analysis in Information Systems Engineering [4].

The tradeoff between expressiveness and complexity makes us deal with the problem of obtaining IKMEs which support real applications. In our opinion, a language expressive enough to represent knowledge captured in every application domain is a vain hope. Conversely, a methodological approach based on formal results and characterized by an engineering perspective is a concrete step towards the diffusion of IKMEs in the industrial practice. This issue is close to the need for a rational management of KR-based support tools already pointed out in [7, 3].

2.1 Formalizing behaviors at the conceptual level

As an example of our proposal at the domain-dependent, user-oriented level, we extend and formalize Data Flow Diagrams (DFDs) introduced in Structured Analysis [11] and used in virtually any approach dealing with behavioral aspects. We capture DFDs by way of the Engineering Domain Description Language ($\mathcal{EDDL}$), which incorporates a DL to express the structure of data and processes. The language discussed in this paper is a significantly enhanced version of the one first proposed in [5]. Notably, the treatment of flow constraints was absent in the original proposal. Capturing (and extending) in an integrated way DFDs together with data models is an important issue in DE. In fact, DFDs represent that behavioral aspect which is considered as relevant in popular approaches dealing with the static component of behaviors (i.e. processes and functions as black boxes), including object-oriented approaches such as OMT [9].

The $\mathcal{EDDL}$ syntax for processes and flows

We use an alphabet $\mathcal{P}$ for processes and two alphabets $\mathcal{C_I}$ and $\mathcal{C_O}$ for input and output channels respectively. The syntax for declaring a process P is

P **has input** I **output** O **;**

I and O are conjunctions of structures, whose syntax is

structure where CH **is** L **:** U C

Such a structure specifies that at least L data (with L $\geq$ 1) and at most U data, all belonging to concept C, flow through channel CH. We impose that CH $\in \mathcal{C_I}$ when the above structure appears in the input of a process description. Similarly, we impose that CH $\in \mathcal{C_O}$ when the above structure appears in the output. We assume that concept C, which describes the properties of flowing data, is expressed in terms of a suitable DL, e.g., CLASSIC. Note that our approach is independent of the chosen general-purpose DL system. We only assume that the empty concept $\perp$ is expressible in that DL.

When a process P is specified, we assume that all channels not explicitly mentioned in its description are absent. This is called Closed Terminology Assumption (CTA) in [10]. Intuitively, for each channel $\textsc{ch} \in \mathcal{C}$ not in in the description of P, statement **structure where** CH **is** 0 **:** 0 $\perp$ is always implicitly added to this description.

The flow declaration syntax between two processes is

flow F **from** P **.** S **to** Q **.** T **;**

where F is the flow name, P is the source process and Q is the target process. Data flow from output channel $\textsc{s} \in \mathcal{C}_\mathcal{O}$ of P to input channel $\textsc{t} \in \mathcal{C}_\mathcal{I}$ of Q.

The $\mathcal{EDDL}$ semantics for processes and flows

All expressions in our $\mathcal{EDDL}$ language can be given a set-theoretic semantics as follows. An interpretation $\mathcal{I}$ is defined as a triple $(\varepsilon[\cdot], \Delta_C, \Delta_P)$, where Δ_C and Δ_P are two disjoint sets of elements and $\varepsilon[\cdot]$ is a mapping. Function $\varepsilon[\cdot]$ assigns to each atomic data concept C a subset $\varepsilon[\mathrm{C}]$ of Δ_C, to each process P an element of Δ_P and to each channel $\textsc{ch} \in \mathcal{C}_\mathcal{I} \cup \mathcal{C}_\mathcal{O}$ a subset $\varepsilon[\textsc{ch}]$ of $\Delta_P \times \Delta_C$. Depending on the adopted DL, the semantics of complex data follows from the $\mathcal{I}$ of atomic concepts.

We denote the cardinality of a set S as $\sharp\mathrm{S}$. Given a process $\mathrm{P} \in \Delta_P$ and a channel CH, we denote with $\varepsilon[\textsc{ch}](\mathrm{P})$ the data inflowing or outflowing P through CH, i.e., the set $\{y \in \Delta_C \mid (\mathrm{P}, y) \in \varepsilon[\textsc{ch}]\}$. The semantics of a structure is thus as follows:

$$\begin{aligned} &\varepsilon[\textbf{structure where } \textsc{ch} \textbf{ is } \textsc{l} : \textsc{u}\ \mathrm{C}] = \\ &\quad = \{ \mathrm{P} \in \Delta_P \mid \textsc{l} \leq \sharp\varepsilon[\textsc{ch}](\mathrm{P}) \leq \textsc{u} \text{ and} \\ &\qquad \text{and } \forall y \in \varepsilon[\textsc{ch}](\mathrm{P}) : y \in \varepsilon[\mathrm{C}] \} \qquad (1) \end{aligned}$$

where for each input structure, $\textsc{ch} \in \mathcal{C}_\mathcal{I}$ as well as, for each output structure, $\textsc{ch} \in \mathcal{C}_\mathcal{O}$. A conjunction of structures, i.e., I_1 **and** I_2 is interpreted as the intersection of interpretations, i.e., $\varepsilon[\mathrm{I}_1] \cap \varepsilon[\mathrm{I}_2]$. A concept C is said to be *satisfiable* if there exists an interpretation $(\varepsilon[\cdot], \Delta_C, \Delta_P)$ such that $\varepsilon[\mathrm{C}] \neq \emptyset$. Let $\mathcal{C} \equiv \mathcal{C}_\mathcal{I} \cup \mathcal{C}_\mathcal{O}$ denote the union of channel alphabets and let $\mathcal{C}_P \subseteq \mathcal{C}$ denote the channel names used in the description of a process P, i.e., P **has input** I **output** O **;** . Interpretation $\mathcal{I}$ satisfies the description of P if

$$\varepsilon[\mathrm{P}] \in \varepsilon[\mathrm{I}] \cap \varepsilon[\mathrm{O}] \bigcap_{\textsc{ch} \notin \mathcal{C}_P} \{ p \in \Delta_P \mid \varepsilon[\textsc{ch}](p) = \emptyset \}$$

Concepts I and O are interpreted as in (1), the term in brackets expresses CTA and $\textsc{ch} \notin \mathcal{C}_P$ means that $\textsc{ch} \in \mathcal{C} - \mathcal{C}_P$. Once written in first-order logic, CTA modifies the specification of P by adding (the finite set of) formulae $\forall y\ (\neg\,\textsc{ch}(\mathrm{P}, y))$ for each $\textsc{ch} \notin \mathcal{C}_P$. This differs from role closure in CLASSIC, which operates on ABox assertions.

We turn now to the interpretation of flow assertions, which is a key feature of this example. An interpretation $\mathcal{I}$ satisfies a flow assertion **flow** F **from** P **.** s **to** Q **.** t **;** from a source process P through its output channel S to a target process Q through its input channel T if all the elements which are output of P through S are input of Q through T. In formulae, $\{x \in \Delta_C \mid (\varepsilon[\mathrm{P}], x) \in \varepsilon[\textsc{s}]\} = \{y \in \Delta_C \mid (\varepsilon[\mathrm{Q}], y) \in \varepsilon[\textsc{t}]\}$.

Finally, an interpretation satisfies a DFD if it satisfies at once all parts of the corresponding structured description, i.e., both process and flow descriptions, as well as any constraint in the adopted DL. We call such an interpretation a *model* of the considered description. We thus say that a description is *satisfiable* if it has a model. The following *DFD Theorem* [4] holds:
A DFD is satisfiable if and only if

1. $\forall$ process description P **has input** I **output** O **;** the two conjunctions of I and O are satisfiable
2. $\forall$ flow assertion **flow** F **from** P **.** s **to** Q **.** t **;**
   (a) source process P contains in its output **structure where** S **is** $\textsc{l}_\textsc{s} : \textsc{u}_\textsc{s}$ C;
   (b) target process Q contains in its input **structure where** T **is** $\textsc{l}_\textsc{t} : \textsc{u}_\textsc{t}$ D;
   (c) cardinalities of source and target channels are mutually compatible, i.e., the two integer intervals $(\textsc{l}_\textsc{s}, \textsc{u}_\textsc{s})$ and $(\textsc{l}_\textsc{t}, \textsc{u}_\textsc{t})$ have the non-empty intersection $(\ max(\textsc{l}_\textsc{s}, \textsc{l}_\textsc{t}), min(\textsc{u}_\textsc{s}, \textsc{u}_\textsc{t})\ )$;
   (d) concept C **and** D is satisfiable.

2.2 Linking the conceptual and epistemological levels

To perform automated analysis of a domain-dependent knowledge base, the user-oriented conceptual level must be linked to the logic-oriented epistemological level, where inferences are drawn from domain-independent concepts. In order to accomplish this task, two different ways of linking the two levels be undertaken. First, following the PROTODL approach originally introduced in [2], specific inferences can be added to a general-purpose DL system and reasoning is completely performed at the epistemological level. In this case, correctness and completeness of the overall set of inferences drawn by the extended system must be proved w.r.t. the semantics of the conceptual level. Second, following the approach originally introduced in [4], a set of emulating concepts is added at the epistemological level. Part of reasoning is directly performed at the conceptual level, while a general-purpose DL system is used at the epistemological level. In this case, to each result of a logical entailment in the conceptual model it corresponds the same result of a different entailment in the epistemological model.

Linking the two levels with PROTODL

Following [2], we can extend a general-purpose DL system with natural-deduction inference rules (see Tabs. 1 and 2) which axiomatize the inferences that the domain-dependent application should draw. These rules have been introduced to deal with the static part of behaviors in $\mathcal{EDDL}$-based conceptual modeling and analysis, taking CTA into account. In the following rules, the connective "$\Rightarrow$" denotes set inclusion. Elements whose details are of no concern for understanding rules are denoted by the underscore (_) symbol. Note that, differently from the PROTODL approach, the above rules are *not* the semantics of our system. Instead, since this is given independently from the rules (see [4]), we must (and can) prove that rules are sound and complete w.r.t. the previously given semantics. Moreover, PROTODL strictly depends on the underlying CLASSIC DL system. This means that no general-purpose DL reasoner other than CLASSIC can be used, thus implicitly constraining the expressive power of the epistemological level.

Linking the two levels with emulators

Emulators are generic concepts (pseudoconcepts) introduced at the domain-independent level in order to reproduce the results of specialized deductive services using generic deductive services. In this way, there is no need of developing specialized inference procedures, which are in any case limited to a specific meta-concept. Moreover, there is not even any need of programming modularly flexible generic reasoners, without knowing the decidability, correctness and completeness properties of the underlying formal calculus. In other words, a minimal set of inferences and a minimal expressive power in terms of concept constructors is needed. This means that different DL reasoners can be used in an interchangeable way. However, differently from the PROTODL case, there is no general mechanism for building emulators. Each time an entailment at the conceptual level has to be emulated by a general-purpose DL system, pseudoconcepts must be developed from scratch.

The usage of emulators introduces an explicit partition and distinction in the way automated reasoning is performed within an IKME. In the general case, including the above PROTODL approach, all reasoning is performed semantically. Given a domain knowledge base at the conceptual level, this is rewritten at the epistemological level in such a way that all epistemological assumptions are satisfied. Conversely, in the emulator-based approach, part of reasoning is performed syntactically. In this way, it is not necessary to overload the translation of a domain knowledge base at the conceptual level with fragments arising from specific epistemological assumptions.

As an exemplary case of the above two different ways of reasoning, we show how processes and flow constraints

Every process with an inconsistent Input/Output is inconsistent

$$\frac{\mathrm{I} \Rightarrow \bot}{\left(\begin{array}{l}\_\ \textbf{has}\\ \textbf{input}\ \mathrm{I}\\ \textbf{output}\ \_\ \textbf{;}\end{array}\right) \Rightarrow \bot}$$

$$\frac{\mathrm{O} \Rightarrow \bot}{\left(\begin{array}{l}\_\ \textbf{has}\\ \textbf{input}\ \_\\ \textbf{output}\ \mathrm{O}\ \textbf{;}\end{array}\right) \Rightarrow \bot}$$

Rules for the CTA

$$\frac{\left(\begin{array}{l}\mathrm{P}\ \textbf{has}\\ \textbf{input}\ \_\\ \textbf{output}\ \mathrm{O}\ \textbf{;}\end{array}\right),\ \textsc{ch} \in \mathcal{C} - \mathcal{C}_P}{\left(\begin{array}{l}\mathrm{P}\ \textbf{has}\\ \textbf{input}\ \_\\ \textbf{output}\ \mathrm{O}\ \textbf{and}\\ \textbf{structure where}\\ \textsc{ch}\ \textbf{is}\ 0\,\textbf{:}\,0\ \bot\ \textbf{;}\end{array}\right)}$$

$$\frac{\left(\begin{array}{l}\mathrm{Q}\ \textbf{has}\\ \textbf{input}\ \mathrm{I}\\ \textbf{output}\ \_\ \textbf{;}\end{array}\right),\ \textsc{ch} \in \mathcal{C} - \mathcal{C}_Q}{\left(\begin{array}{l}\mathrm{Q}\ \textbf{has}\\ \textbf{input}\ \mathrm{I}\ \textbf{and}\\ \textbf{structure where}\\ \textsc{ch}\ \textbf{is}\ 0\,\textbf{:}\,0\ \bot\\ \textbf{output}\ \_\ \textbf{;}\end{array}\right)}$$

Table 1: Sequent rules for processes and CTA

can be emulated using the CLASSIC language. Let all the channel names in the DKB be $\mathcal{C}$={ R, S, T }, and let us consider two process descriptions and a flow assertion expressed at the conceptual level (again, symbol "_" stands for a structure whose details are of no concern):

P **has input** _ **output structure where** S **is** 2**:**∞ C **;**
Q **has input structure where** T **is** 1**:**7 D **output** _ **;**
flow F **from** P **.** S **to** Q **.** T **;**

In CLASSIC, we can use individuals `P` and `Q` to emulate processes, roles `r,s,t` for channels, concept `FLOW` and role `flow` to emulate the flow assertion.

If fully semantical reasoning is performed, specific concept constructors must be added to satisfy CTA. Notably, these constructors must assert that all channels which are not explicitly part of a process definition have to be considered as semantically absent. In practice, this means that their cardinality is at most 0 in our example. We can thus write

A flow assertion is inconsistent when
number restrictions of input and output are incompatible

$$\frac{\begin{array}{c}\left(\begin{array}{l}\text{P } \textbf{has}\\ \textbf{input \_}\\ \textbf{output \_ and}\\ \textbf{structure where}\\ \text{S } \textbf{is } L_S : U_S \text{ \_ ;}\end{array}\right), \\ \left(\begin{array}{l}\text{Q } \textbf{has}\\ \textbf{input \_ and}\\ \textbf{structure where}\\ \text{T } \textbf{is } L_T : U_T \text{ \_}\\ \textbf{output \_ ;}\end{array}\right), \\ (L_S, U_S) \cap (L_T, U_T) = \emptyset\end{array}}{\left(\begin{array}{l}\textbf{flow \_}\\ \textbf{from } \text{P . S}\\ \textbf{to } \text{Q . T ;}\end{array}\right) \Rightarrow \bot}$$

A flow assertion is inconsistent when
the intersection of exchanged data is empty

$$\frac{\begin{array}{c}\left(\begin{array}{l}\text{P } \textbf{has}\\ \textbf{input \_}\\ \textbf{output \_ and}\\ \textbf{structure where}\\ \text{S } \textbf{is \_ : \_ } \text{X ;}\end{array}\right), \\ \left(\begin{array}{l}\text{Q } \textbf{has}\\ \textbf{input \_ and}\\ \textbf{structure where}\\ \text{T } \textbf{is \_ : \_ } \text{Y}\\ \textbf{output \_ ;}\end{array}\right), \\ \text{X } \textbf{and } \text{Y} \Rightarrow \bot\end{array}}{\left(\begin{array}{l}\textbf{flow \_}\\ \textbf{from } \text{P . S}\\ \textbf{to } \text{Q . T ;}\end{array}\right) \Rightarrow \bot}$$

Table 2: Sequent rules for flows

```
( individual P    ( and _ ( all s C ) ( atleast 2 s )
                  ( atmost 0 r ) ( atmost 0 t ) ) )
( individual Q    ( and _ ( all t D ) ( atleast 1 t )
                  ( atmost 7 t ) ( atmost 0 r )
                  ( atmost 0 s ) ) )
( conceptdef F    ( and ( all flow ( and C D ) )
                  ( atleast 1 flow ) ( atleast 2 flow )
                  ( atmost 7 flow ) ) )
```

Note that CTA is emulated (i) in `P` by the two fragments (**atmost** 0 r) and (**atmost** 0 t) and (ii) in `Q` by the two fragments (**atmost** 0 r) and (**atmost** 0 s).

If partially syntactical reasoning is performed, no kind of specifical concept constructor must be added to satisfy CTA. It is sufficient to check that all channels which are part of the flow constraint really belong to the processes linked by the flow itself. In both cases, we can prove that to check whether the process descriptions and the flow assertion are satisfiable is sufficient to check whether the individuals `P` and `Q` and the concept `FLOW` are satisfiable.

References

[1] S. Bergamaschi, S. Lodi, and C. Sartori. The E/S Knowledge Representation System. *Data & Knowledge Engineering*, 14:81–115, 1994.

[2] A. Borgida. Towards the Systematic Development of Description Logic Reasoners: CLASP reconstructed. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 259–269, 1992.

[3] A. Borgida and R. J. Brachman. Customizable Classification Inference in the ProtoDL Description Management System. In *Proc. of the Int. Conf. on Information and Knowledge Management (CIKM'92)*, 1992.

[4] E. Compatangelo, F. M. Donini, and G. Rumolo. Reasoning with behavioural knowledge in application domain models. In *Proc. of the 5th Conf. of the Italian Association for Artificial Intelligence (AI*IA'97)*, volume 1321 of *Lecture Notes In Artificial Intelligence*, pages 369–380. Springer-Verlag, 1997.

[5] E. Compatangelo and G. Rumolo. $\mathcal{EDDL}_{DP} + \mathcal{TDDL}_{DP}$ = double-level approach to Domain Knowledge Modelling. In *Information Modelling and Knowledge Bases VIII*, pages 279–295. IOS Press, 1997.

[6] P. Devambu et al. LASSIE: A Knowledge-Based Software Information System. *Communications of the ACM*, 34(5):36–49, 1991.

[7] J. Doyle and R. S. Patil. Two theses of knowledge representation: language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48:261–297, 1991.

[8] R. Elmasri and S. Navathe. *Fundamentals of Database Systems, 2nd ed.* Benjamin-Cummings, 1994.

[9] J. Rumbaugh et al. *Object-Oriented Modelling and Design*. Prentice-Hall, 1991.

[10] R. A. Weida. Closed Terminologies in Description Logics. In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI'96)*, pages 592–599, 1996.

[11] E. Yourdon. *Modern Structured Analysis*. Prentice-Hall, 1989.